Mikko Ojala

Sähkön pörssihintojen liittäminen IoT-TICKET-tuotteeseen

Opinnäytetyö
Syksy 2019
SeAMK Tekniikka
Automaatiotekniikan tutkinto-ohjelma

SeAMK
SEINÄJOEN AMMATTIKORKEAKOULU
SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

SEINÄJOEN AMMATTIKORKEAKOULU

Opinnäytetyön tiivistelmä

Koulutusyksikkö: Tekniikan yksikkö

Tutkinto-ohjelma: Automaatiotekniikka

Suuntautumisvaihtoehto: Sähköautomaatio

Tekijä: Mikko Ojala

Työn nimi: Sähkön pörssihintojen liittäminen IoT-Ticket-tuotteeseen

Ohjaaja: Ismo Tupamäki

Vuosi: 2019 Sivumäärä: 41

Opinnäytetyö toteutettiin Wapice Osakeyhtiölle. Yritys tarjoaa erilaisia palveluita, jotka liittyvät analyyseihin, tekoälysovelluksiin ja big datan käsittelyyn. Lisäksi yrityksen toimenkuvaan kuuluvat erilaiset palvelumuotoilut, digitalisaatioon liittyvät web- ja mobiiliratkaisut.

Tämän opinnäytetyön tavoitteena oli tehdä Wapicen IoT-TICKET-tuotteeseen lisäpalvelu. Palvelu hakee sähkön pörssihinnat Nord Poolin SFTP-serveriltä ja lähettää hinnat IoT-TICKET-palveluun sen tilanneille asiakkaille.

Tuloksena työstä saatiin toimiva sovellus, joka voidaan ottaa käyttöön IoT-TICKET-palvelussa. Alun perin mikropalvelusovellukseksi suunniteltu sovellus muuttui työn aikana monoliittisovellukseksi.

Avainsanat: IoT, Java Spring, IoT-TICKET, Teollinen internet

SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Thesis abstract

Faculty: School of Technology

Degree programme: Automation Engineering

Specialisation: Electricity Automation

Author/s: Mikko Ojala

Title of thesis: Bringing electricity stock prices to an IoT-Ticket product

Supervisor: Ismo Tupamäki

Year: 2019 Number of pages: 41

The commissioner of this thesis was Wapice Oy. The company offers different kind of services related to analytics, artificial intelligence and big data handling. The company also offers service design, and web- and mobile-solutions related to digitalization.

The purpose of the thesis was to make an extra service to the IoT-TICKET. The service gets electricity stock prices from the Nord Pool SFTP-server and then sends the prices to the IoT-TICKET for customers who have ordered this service.

As the result of the thesis, there is a working program ready to be used on the IoT-TICKET platform. The program was first designed to be a micro service program, but during the development project, it changed to a monolith program.

Keywords: IoT, Java Spring, IoT-TICKET, Industrial Internet

SISÄLTÖ

Kuva- ja taulukkoluettelo

Käytetyt termit ja lyhenteet

4G	Nimitys neljännen sukupolven langattomista tiedonsiirtoteknologioista.
5G	Nimitys viidennen sukupolven langattomista tiedonsiirtoteknologioista.
Annotaatio	Annotaatio on Java-ohjelmointikielen tapa lisätä synteettistä metadataa luokkiin, metodeihin, muuttujiin ja parametreihin.
AOP	Aspect Oriented Programming, suomeksi aspektiohjelmointi, on ohjelmointimalli.
API	Application programming interface, suomeksi ohjelmointirajapinta, määrittää ohjelmiston tavan kommunikoida muiden ohjelmistojen kanssa.
Autonominen tuote	Itsestään toimiva tuote, joka toimii ilman ihmisen apua.
Dashboard	Visuaalinen raportointityökalu.
Datatag	IoT-TICKET-palvelussa tallennetaan kerätty data datatageihin.
Depency Injection	Suomeksi riippuvuusinjektio on Spring-ohjelmointikielen tapa, jolla riippuvuus annetaan.
IDE	Integrated development environment, suomeksi ohjelmointiympäristö.
Docker	Konttisovellus, joka pakkaa ajettavan sovelluksen ja sen kaikki riippuvuudet yhteen pakettiin eli konttiin.
Git	Hajautettu versionhallintaohjelmisto.

Instanssipalvelin	Mikropalveluissa käytetty palvelin, joka pitää kirjaa ja välittää eri mikropalvelinten nimet, osoitteet ja portit eteenpäin.
Luokka	Olio-ohjelmoinnissa luokka määrittelee olion ominaisuudet.
Metodi	Olio-ohjelmoinnissa olion osa, joka suorittaa jonkin halutun tehtävän.
Modulaarinen	Itsenäisistä osista koostuva kokonaisuus.
MQTT	Message Queuing Telemetry Transport on viestintäprotokolla, joka perustuu julkaisija-tilaaja-periaatteeseen.
Muuttuja	Nimetty tietovarasto ohjelmoinnissa.
Palvelumuotoilu	Palvelujen kehittämistä muotoilussa käytetyin menetelmin.
REST	Representational State Transfer on HTTP-protokollaan perustuva ohjelmointirajapinta.
Regex	Suomeksi säännöllinen lauseke on lauseke, joka määrittelee säännöllisen joukon merkkijonoja.
RSA	Rivest-Shamir-Adleman on salausalgoritmi, joka käyttää julkista avainta.
SFTP	SSH File Transfer Protocol on protokolla tiedostojen siirtämiseen SSH-protokollan yli.

1 JOHDANTO

1.1 Työn tausta

Sähkön pörssihinta saattaa vaihdella paljon päivän aikana. Ei ole mitenkään harvinaista, että maksimihinta on jopa kaksinkertainen minimihintaan verrattuna. Tämän vuoksi olisi hyvä käyttää suuritehoisia laitteita silloin, kun sähkö on halvimmillaan.

Nord Pool on vuonna 1993 perustettu pohjoismaissa ja Baltian maissa toimiva sähköpörssi. Sen palveluihin kuuluvat muun muassa sähkön kauppa ja tietopalvelu seuraavan päivän sähkön hinnasta. (Nord Pool group[Viitattu 7.8.2019].) Tätä palvelua käytetään tässä opinnäytetyössä syntyneessä sovelluksessa. Nord Poolin omistaa kantaverkkoyhtiöt Statnett SF, Svenska kraftnät, Fingrid Oy, Energinet.dk, Elering, Litgrid and Augstsprieguma tikls (AST) (Nord Pool group[Viitattu 7.8.2019]).

Wapicen IoT-TICKET on IoT-alusta, jolla on mahdollista visualisoida ja analysoida dataa sekä paljon muutakin. Sillä on myös mahdollista ohjata siihen kytkettyjä järjestelmiä. Ajatuksena on tuoda sähkön pörssihinnat IoT-TICKET-tuotteeseen ja siellä hyödyntää niitä esimerkiksi ohjaamaan suuritehoisia laitteita toimimaan silloin, kun sähkö on halpaa.

1.2 Työn tavoite

Työn tavoite oli luoda Wapicen IoT-TICKET-tuotteeseen lisäominaisuus, joka hakee sähkön pörssihinnat Nord Poolin SFTP-serveriltä, kerää hinnat ladatusta tiedostosta ja lähettää hinnat IoT-TICKET-palveluun datatageina ominaisuuden tilanneille asiakkaille.

1.3 Työn rakenne

Luvussa kaksi käsitellään työhön liittyvää teoriaa. Luvussa käsitellään arkkitehtuurimalleja, DevOpsisia, Springiä sekä sen lisäosia, olio-ohjelmointia ja teollista internetiä.

Luvussa kolme käsitellään lisää työhön liittyvää teoriaa. Luvussa käsitellään Wapicen IoT-TICKET-tuotetta, versionhallintaa ja dokumenttien laadintaa.

Luvussa neljä käsitellään työn toteutuksen eri vaiheet. Luvusta löytyy yksityiskohtaista tietoa sovelluksen toteutuksesta, sekä yleisiä esimerkkejä.

Luvussa viisi käsitellään työn tuloksia, valmiin sovelluksen ominaisuuksia sekä sovelluksen tilannetta. Luvussa mietitään myös jälkikehitystä.

Luvussa kuusi mietitään työn onnistumista ja jälkikehitystä.

1.4 Wapice Oy

Wapice on vuonna 1999 perustettu ohjelmistoyritys. Sen palveluihin kuuluvat erilaiset analytiikkaan, tekoälyyn ja pilvipalveluihin kuuluvat sovellukset. Lisäksi yritys toimii konsultoinnin, tietoturvan, DevOpsin ja elektroniikkasuunnittelun alueella. Myös esineiden internetpalvelut, palvelumuotoilu, teknologia- ja digitalisaatioratkaisut sekä web- ja mobiiliratkaisut ovat yrityksen toimialaa. (Wapice [Viitattu 31.7.2019].)

Wapice tekee sekä ohjelmistojen alihankintaa että omia tuotteita. Wapicella on seuraavat omat tuotteet: IoT-TICKET, Summium CPQ, Summium Selector, Wapice Energy Services ja CANrunner. (Wapice [Viitattu 31.7.2019].)

Wapice tekee myös elektroniikkasuunnittelua. Yrityksellä on oma tiedonkeruulaite nimeltä WRM247+. WRM247+ kerää dataa ja lähettää sen IoT-TICKET-järjestelmään. WRM247+-laiteella voi myös ohjata siihen kytkettyjä järjestelmiä. (IoT-TICKET-alusta [Viitattu 31.7.2019].)

2 OHJELMOINTI JA OHJELMARATKAISUT

Tässä luvussa käydään läpi erilaisia arkkitehtuurimalleja, DevOpsia, teollista internetiä, sekä erilaisia ohjelmointitapoja. Luvussa käsitellään myös työssä käytettyä ohjelmointikieltä ja sen komponentteja.

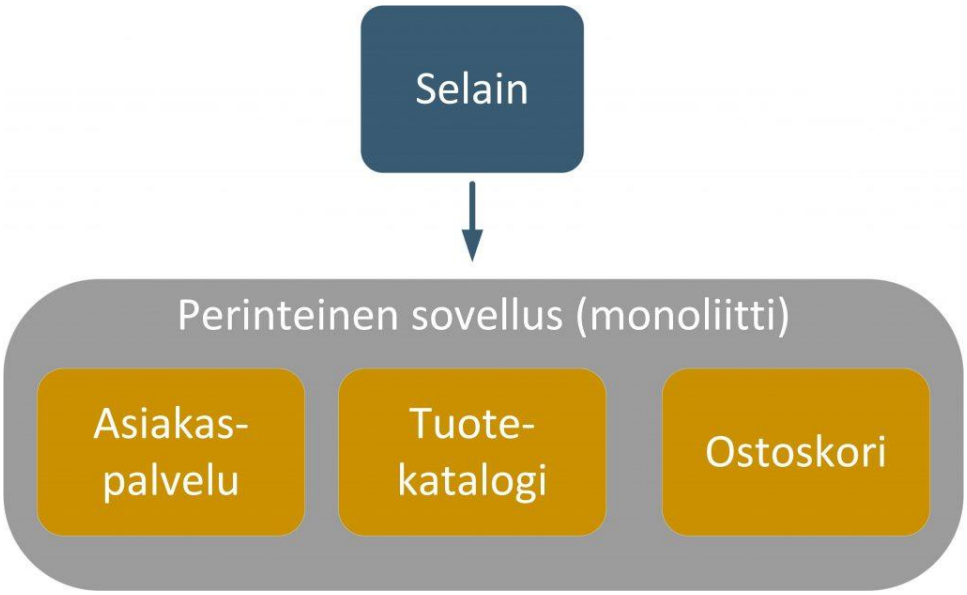
2.1 Arkkitehtuurimallit

Tässä luvussa käsitellään erilaisia arkkitehtuurimalleja. Luvussa käsitellään myös DevOpsia ja sen vaikutusta arkkitehtuurimalleihin.

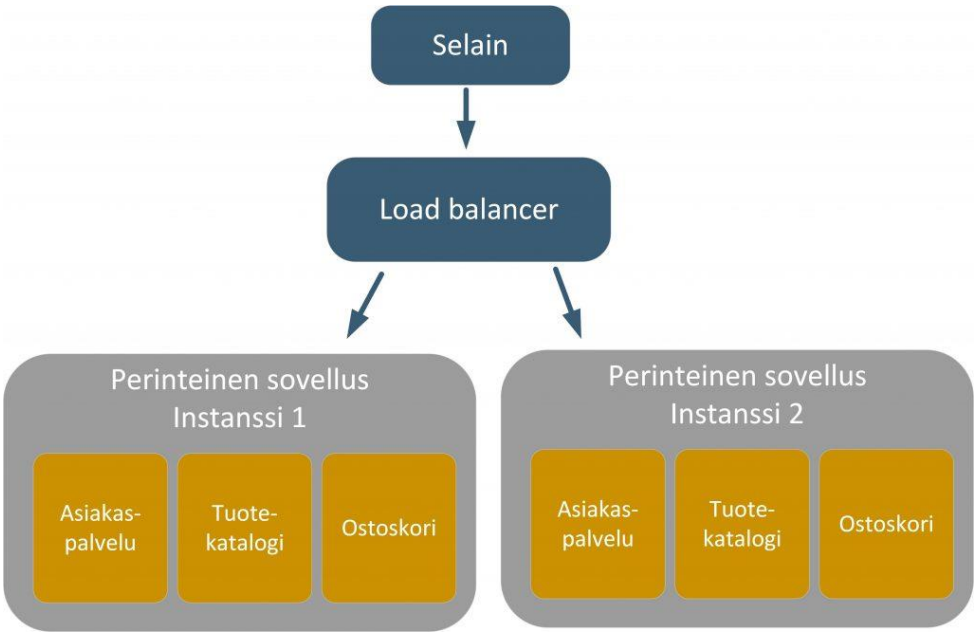
2.1.1 Monoliitti

Monoliittisessa arkkitehtuurimallissa ohjelmisto on yksi iso kokonaisuus. Monoliittisen arkkitehtuurimallin heikkouksia ovat skaalautuvuus, ohjelmiston päivittäminen, ketteryys ja isoissa ryhmissä kehittäminen. Jos halutaan muuttaa jotain ominaisuutta ohjelmistossa, on koko ohjelmisto tunnettava. Lisäksi muutoksessa koko ohjelman rakennetta täytyy muuttaa, ei vain osaa. (Rajesh 2016.)

Monoliittisen arkkitehtuurimallin hyviä puolia ovat järjestelmän selkeys ja testaamisen helppous (Rajesh 2016). Kuvista 1 ja 2 selviää monoliittisen sovelluksen rakenne ja skaalaus.



Kuva 1. Monoliittisovelluksen rakenne (Wallenius [Viitattu 9.8.2019]).



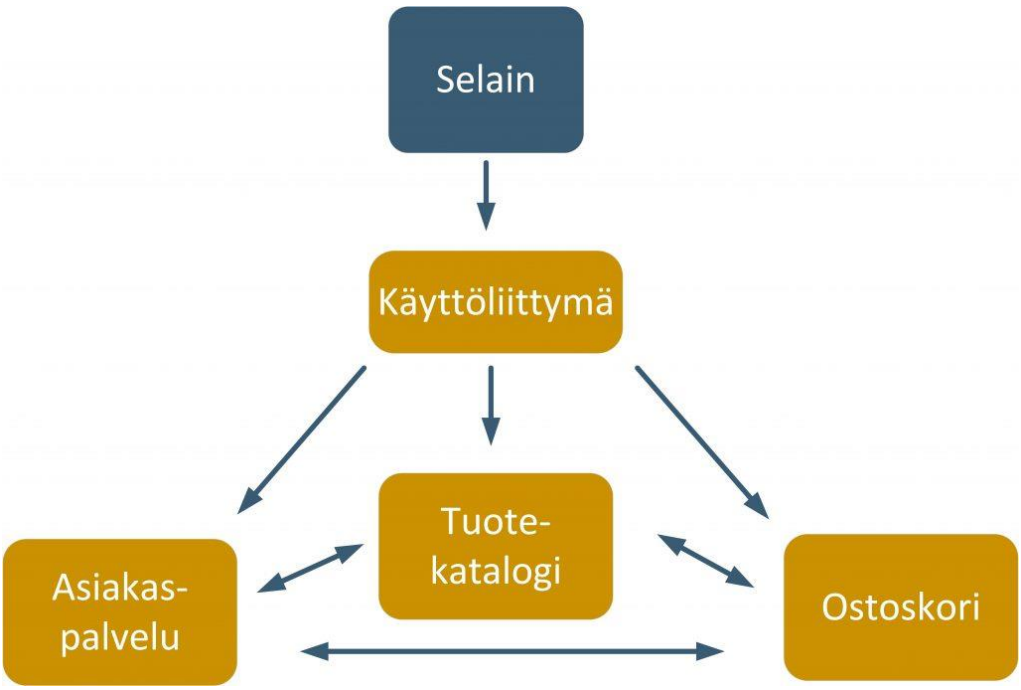
Kuva 2. Monoliittisovelluksen skaalaus (Wallenius [Viitattu 9.8.2019]).

2.1.2 Mikropalvelu

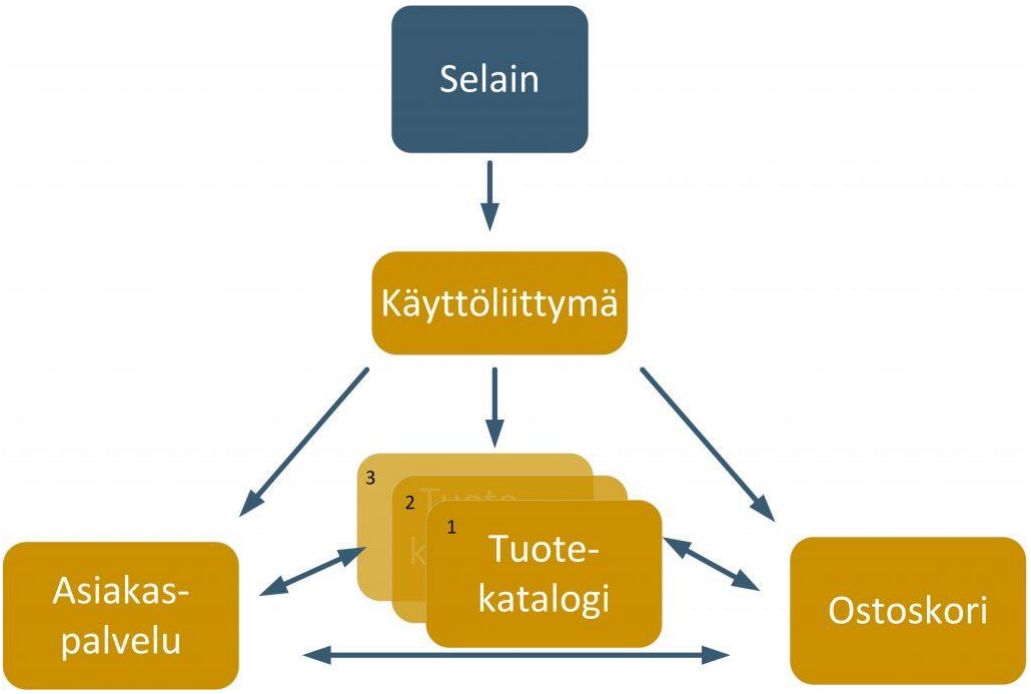
Mikropalvelut ovat arkkitehtuurimalleja, joissa monimutkaisia järjestelmiä hajautetaan itsenäisiksi prosesseiksi. Prosessit kommunikoivat keskenään muodostaen isomman kokonaisuuden. Arkkitehtuurimallin hyviä puolia ovat ketteryys, skaalautuvuus ja kehityksen nopeus. Huonona puolena on järjestelmän monimutkaisuus sekä mahdollinen testaamisen hankaluus. Mikropalvelussa jokainen toiminto on oma palvelunsa. Tämä johtaa joustavaan skaalautumiseen. Ohjelmistossa voi olla jokin

tietty ominaisuus, jota kuormitetaan huomattavasti enemmän kuin muita ominaisuuksia. Tämä asia voidaan ratkaista käynnistämällä ominaisuuden tuottava mikropalvelu useampaan kertaan. Näin saadaan tasoitettua kyseisen ominaisuuden kuormitusta. Monoliittisessa palvelussa tässä tilanteessa joudutaan koko sovellus käynnistämään moneen kertaan, ja tämä aiheuttaa turhaa resurssien kulumista, koska vain yksi ominaisuus on kuormittunut. (Rajesh 2016.)

Kuvista 3 ja 4 selviää mikropalvelusovelluksen rakenne ja skaalaus.



Kuva 3. Mikropalvelusovelluksen rakenne (Wallenius [Viitattu 9.8.2019]).

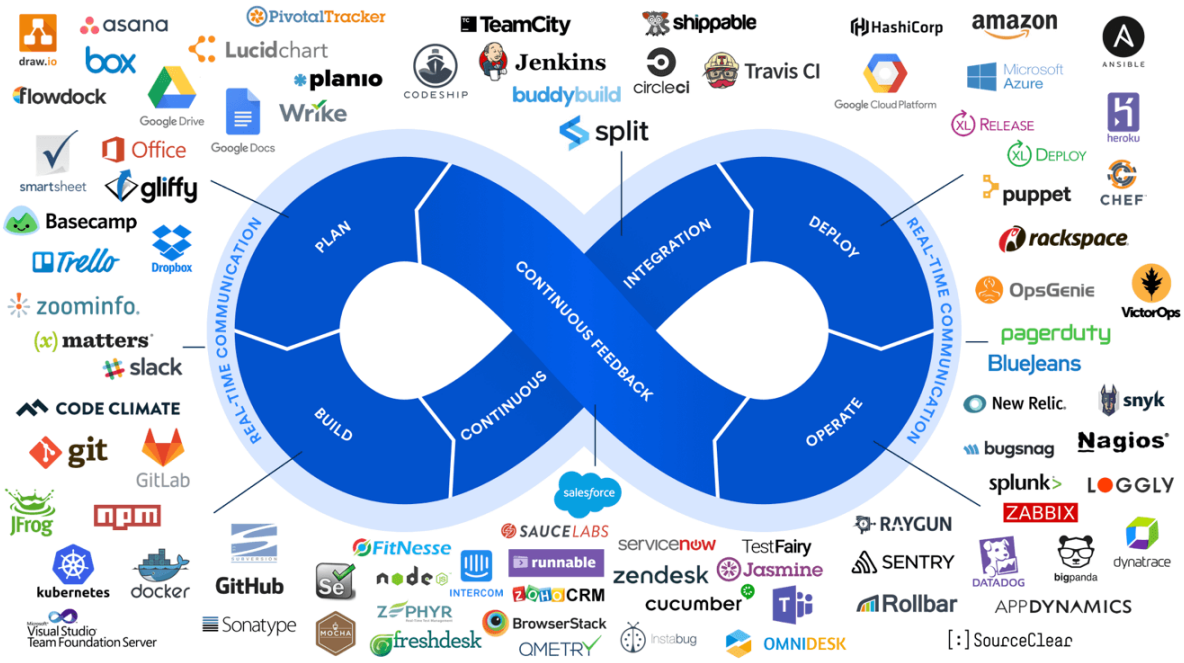


Kuva 4. Mikropalvelusovelluksen skaalaus (Wallenius [Viitattu 9.8.2019]).

2.2 DevOps

DevOps on toimintamalli, joka pyrkii automatisoimaan ohjelmiston julkaisuun ja ohjelmiston ylläpitoon liittyvän kehityksen. DevOpsissa automatisoidaan esimerkiksi paketointi, laadunvarmistus ja julkaisu. DevOps ei pelkästään sisällä teknisiä ratkaisuja, vaan myös organisaatioiden ja ihmisten toimintatapojen muuttamista. (Klementti 2013.)

DevOpsin suurimmat hyödyt ovat nopeat julkaisut ja nopeat vianselvitykset. (Klementti 2013). Joitain suosittuja DevOps-ohjelmistoja ovat Git, Jenkins, Bamboo, Docker, Kubernetes ja Nagios. Tässä työssä käytettiin Git- ja Docker-ohjelmistoja. Kuvassa 5 näkyy erilaisia DevOpsiin liittyviä ohjelmistoja.

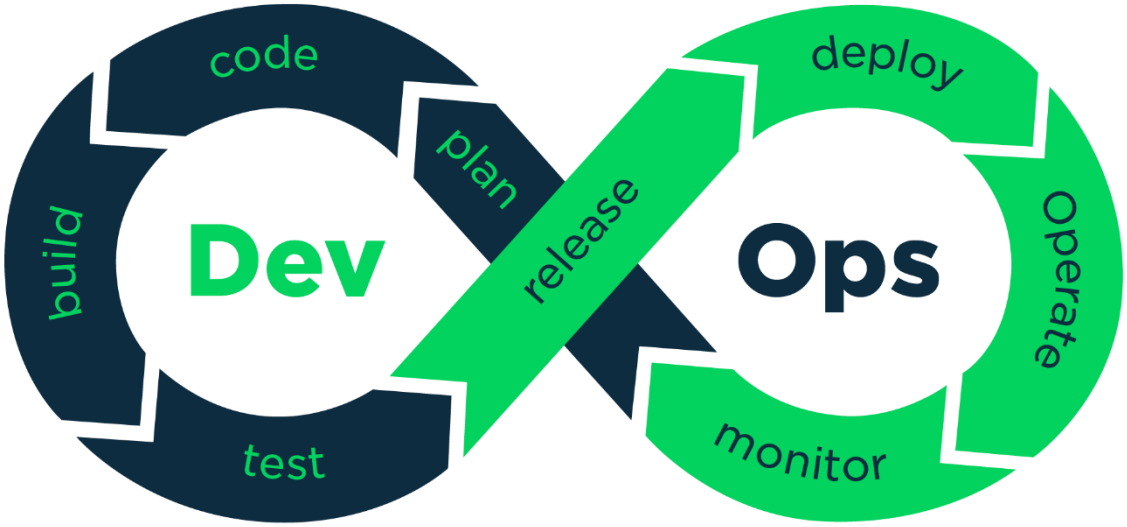


Kuva 5. DevOps-ohjelmistoja (Governor 2017).

Mikropalveluita ajetaan usein Docker-konteissa. Kontit ovat siirreltäviä, itsenäisesti hallittavia ja kevyitä ajoympäristöjä. Jokaisesta mikropalvelusta luodaan oma kontti ja niitä voidaan käynnistää kuormituksen mukaan lisää. Mikropalvelut voivat myös olla käytössä eri palvelimilla. (Rajesh 2016.)

Mikropalvelut voivat olla eri ohjelmointikielillä kirjoitettuja. Ne voivat olla myös saman teknologian eri versioilla kirjoitettuja. Esimerkiksi mikropalvelu A on kirjoitettu Java 1.7 -versiolla, mikropalvelu B Java 1.8 -versiolla ja mikropalvelu C Python 3.7 -versiolla. Vaikka ohjelmoinnissa käytetään erilaisia tekniikoita, toimivat ohjelmat toistensa kanssa, muodostaen toimivan kokonaisuuden. (Rajesh 2016.)

Yksi mikropalvelujen hyvä puoli on se, että ne mahdollistavat DevOps-toimintamallin käytön. DevOpsissa tarkoituksena on pääasiassa nopeuttaa tuotteen toimitusta ja ketteryyttä. Isot monoliittiohjelmistot eivät ole DevOps-ystävällisiä. Monet DevOps-työkalut perustuvat nimenomaan mikropalvelujen käytölle. (Rajesh 2016.)



Kuva 6. DevOps (Kornilova 28.4.2017).

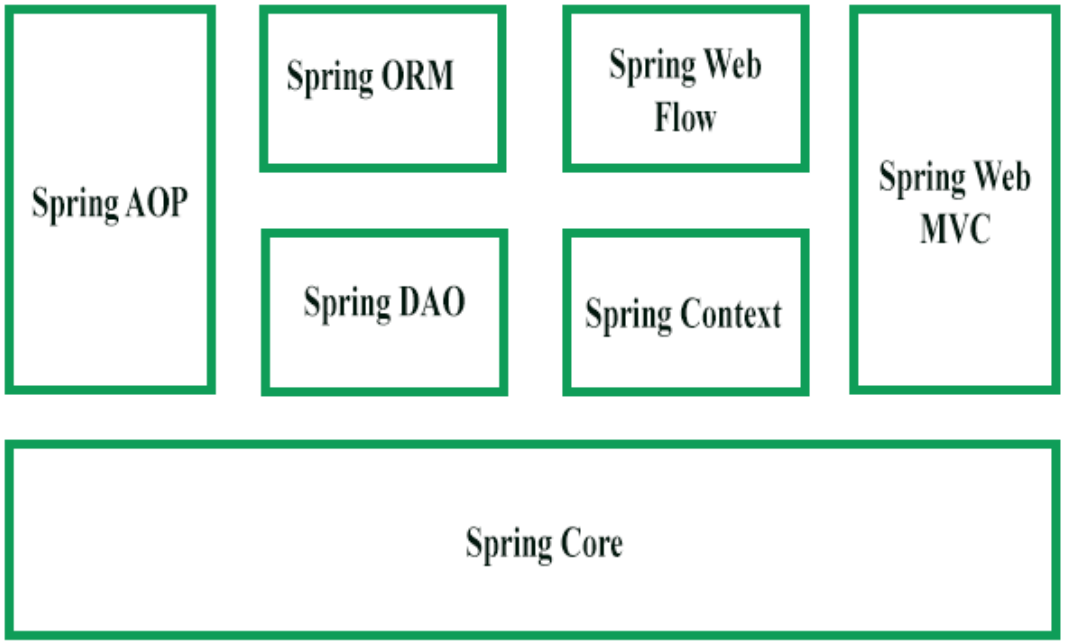
2.3 Spring ja Spring Boot

Tässä luvussa käsitellään Java-kielen Spring ja Spring Boot -kehyksiä. Tässä luvussa käsitellään myös Spring Cloud Netflix -komponentteja.

2.3.1 Spring

Spring on avoimen lähdekoodin Java-kielen kehys. Se on kevyt, perusversio springistä on noin 2 MB:n kokoinen. Se on erittäin suosittu kehys, sitä käyttävät miljoonat kehittäjät ja se on suosituin Java-kehys. Sen hyviä puolia ovat mm:

- testauksen helppous
- modulaarisuus
- riippuvuusinjektio
- AOP (Aspect Oriented Programming)
- kehittämisen nopeus. (Spring Framework [Viitattu 9.8.2019].)



Kuva 7. Spring-arkkitehtuuri (Introduction to Spring Framework [Viitattu 9.8.2019])

2.3.2 Spring Boot

Spring Boot on Springin lisäkomponentti, jonka avulla on helppo paketoida sovellus itsenäiseksi ja tuotantovalmiiksi kokonaisuudeksi. Sen hyviä puolia on muun muassa:

- helppous ymmärtää ja kehittää spring-sovelluksia
- tuottavuuden lisäys
- kehittämiseen kuluvan ajan väheneminen. (Spring Boot [Viitattu 9.8.2019].)

2.3.3 Spring Cloud Netflix

Spring Cloud Netflix tarjoaa Spring Boot -sovelluksiin Netflix-komponentteja. Netflix-komponentit hoitavat mikropalveluarkkitehtuuriin liittyviä tehtäviä. Spring Cloud Netflixiin kuuluvat Eureka, Hystrix, Zuul, Ribbon, Config Server, OAuth2. (What is Micro Service [Viitattu 9.8.2019].) Taulukosta 1 selviää komponenttien tehtävät.

Taulukko 1. Netflix-komponentit (What is Micro Service [Viitattu 9.8.2019]).

Komponentti	Tehtävä
Eureka	instanssipalvelin
Zool	reunapalvelin
Config Server	keskitetty konfiguraatiopalvelin
Ribbon	dynaaminen reititys ja kuormantasaus
OAuth2	Oauth 2.0 suojatut API:t
Hystrix	monitorointi

Komponenteista tärkein on Eureka, joka toimii instanssipalvelimena pitäen kirjaa käytössä olevista mikropalveluista sekä niiden osoitteista ja porteista. Jokainen mikropalvelu rekisteröityy Eurekaan, joka välittää tietoja eteenpäin saatavilla olevista mikropalveluista. (What is Micro Service [Viitattu 9.8.2019].)

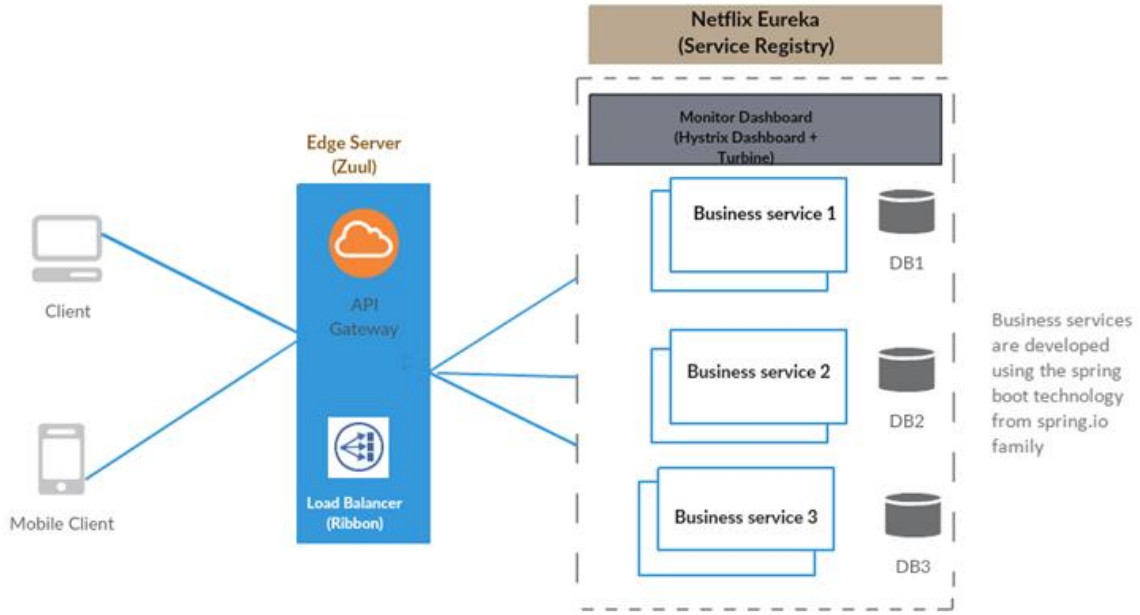
Yksinkertaisimmillaan mikropalvelusovellus sisältää instanssipalvelimen ja mikropalveluita. Käytännössä käytössä on usein myös muita komponentteja.

Ribbon käyttää Eurekasta saatavaa tietoa. Jos yhtä instanssia löytyy enemmän kuin yksi kappale, niin Ribbon käyttää kuorman tasausta kyseisille komponenteille. (What is Micro Service [Viitattu 9.8.2019].)

Zuul on yhteydessä ulkomaailmaan ja hoitaa mm. tunnistautumisia, kuorman tasausta, tietoturvaa ja dynaamista reititystä. Zuul käyttää Ribbonia selittäkseen saatavilla olevat palvelut ja ohjaa ulkoiset pyynnöt oikeaan palveluun. (What is Micro Service [Viitattu 9.8.2019].)

Hystrixin avulla voidaan tarkkailla palveluita ja parantaa niiden virheensietoa. Hystrix lisää tietovirran- ja katkaisijan ominaisuudet palveluun. (Introduction to Hystrix [Viitattu 9.8.2019].)

Kuvasta 8 selviää Spring Cloud Netflixillä tehdyn mikropalvelusovelluksen arkkitehtuuri.



Kuva 8. Spring Cloud Netflix -arkkitehtuuri (What is Micro Service [Viitattu 9.8.2019])

2.4 Olio-ohjelmointi

Olio-ohjelmointi on ohjelmointimalli, jolla kuvataan järjestelmien rakennetta ja toimintaa. Ennen olio-ohjelmointia oli käytössä proseduraalinen ohjelmointimalli, joka perustui pelkästään aliohjelmien käyttöön. Proseduraalisessa ohjelmoinnissa koodin uudelleenkäyttö, ylläpidettävyys ja laajennettavuus ovat ongelmia, joihin olio-ohjelmointi tuo parannuksia. Toisaalta taas olio-ohjelmoinnissa suunnitteluvaihe kestää kauemmin. Proseduraalisessa ohjelmoinnissa pienten sovellusten teko onnistuu hyvin, mutta isojen kohdalla olio-ohjelmoinnin hyödyt nousevat esiin. (Koskimies 2000, 21-25.)

Oliot sisältävät toiminnallisen puolen, joka koostuu metodeista ja tietoa säilyttävän puolen, joka koostuu attribuuteista. Kun olio luodaan, siihen syntyy tunniste eli viite, joka on uniikki jokaisella oliolla. (Koskimies 2000, 30.)

Oliot määritellään tavallisesti luokassa. Olio on luokan ilmentymä. Luokassa määritellyt attribuutit ja metodit ovat samat jokaisella siitä luodulla oliolla. Jokaisella oliolla voi olla kuitenkin eri attribuuttien arvo. (Koskimies 2000, 37.)

Esimerkiksi opiskelija voidaan mallintaa luokkana. Opiskelijalla voisi olla attribuutit: etunimi, sukunimi, ikä ja osoite. Metodina voisi olla vanhene, joka lisäisi ikään yhden vuoden. Näin ollen voitaisiin luoda eri opiskelijoista olioita käyttäen samaa luokkaa.

Kuvassa 9 on esimerkki Opiskelija-luokasta Java-kielellä toteutettuna.

```
3  public class Opiskelija {
4
5      String etunimi;
6      String sukunimi;
7      int ika;
8      String osoite;
9
10     public Opiskelija(String etunimi, String sukunimi, int ika, String osoite) {
11         this.etunimi = etunimi;
12         this.sukunimi = sukunimi;
13         this.ika = ika;
14         this.osoite = osoite;
15     }
16
17     public void vanhene() {
18         ika=ika+1;
19     }
20
21 }
22
```

Kuva 9. Opiskelija-luokka Java-kielellä

Kuvassa 10 on esimerkki, miten kyseistä luokkaa voitaisiin käyttää Java-ohjelmoinnissa.

```
18         Opiskelija opiskelija1= new Opiskelija("Matti","Meikäläinen",21,"Tuntematonkatu 1");
19         Opiskelija opiskelija2= new Opiskelija("Maija","Meikäläinen",24,"Tuntematonkatu 1");
20
21         opiskelija1.vanhene();
22         opiskelija2.vanhene();
23
```

Kuva 10. Opiskelija-luokan käyttö Java-kielellä

Luokassa täytyy olla myös rakentaja, että olio voidaan luoda. Rakentajassa asetetaan attribuutteihin sopivat arvot. (Koskimies 2000, 41).

Yksi olioiden tärkeä ominaisuus on periytyminen. Periytymisessä luokka B perii piirteet toiselta luokalta A. Tällöin luokalla B on käytössä kaikki luokan A piirteet. Sillä on myös omia piirteitä. Luokka voi periytyä myös useamman luokan kautta. Esimerkiksi A-luokan perii luokka B, ja B-luokan perii luokka C. Tällöin luokalla C on käytössä kaikki piirteet luokasta A ja B. (Koskimies 2000, 68.)

Esimerkiksi luokka henkilö sisältää attribuutit nimi ja ikä. Luokka opiskelija perii luokan henkilö ja sisältää näin attribuutit nimi ja ikä. Lisäksi luokka opiskelija sisältää attribuutin keskiarvo.

On olemassa myös moniperiytymistä, mutta kaikki oliokielet eivät tue moniperintää. Moniperinnässä yksi luokka perii kahden eri luokan ominaisuudet. (Koskimies 2000, 80.)

2.5 Teollinen internet

Teollinen internet tarkoittaa teollisuuden digitalisointia paikoissa, joissa sitä ei ole ennen hyödynnetty. Tämän on mahdollistanut muun muassa halvemmat, pienemmät ja tarkemmat anturit sekä kehittyneet internetyhteydet. Varsinkin langattomien internetyhteyksien kehitys on ollut suuri tekijä. Nykyiset 4G-yhteydet ovat jo varsin nopeita ja niillä on hyvä kuuluvuusalue. Tulevien 5G-yhteyksien ennustetaan tuovan todella huomattavia etuja teolliselle internetille. (Collin & Saarelainen 2016.)

Kerättävän datan yleisimmät käyttötavat ovat vikatilanteisiin liittyvät hälytykset ja etävalvonta reaaliaikaisesti. Kuitenkin yritykset hyötyisivät eniten datan käyttämisestä tuotteiden optimointiin ja vikojen ennustamiseen. (Collin & Saarelainen 2016.)

Teollisen internetin sovellusalueita voi keksiä lähes loputtomiin. Seuraavat neljä ovat jo käytössä olevia sovellusalueita:

1. Etävalvonta, etähallinta, optimointi ja etäpäivitykset
2. Ennakoiva huoltopalvelu ja analytiikka
3. Uusi datapohjainen palveluliiketoiminta
4. Älykäs tehdas ja autonomiset tuotteet. (Collin & Saarelainen 2016.)

Teollisessa internetissä yleisesti käytettyjä standardeja ovat mm. Modbus, CoAP, MQTT, AMQP, XMPP, DDS, OPC UA, MTConnect ja AutomationML (Collin & Saarelainen 2016).

3 WAPICE IOT-TICKET, VERSIONHALLINTA JA DOKUMENTAATIO

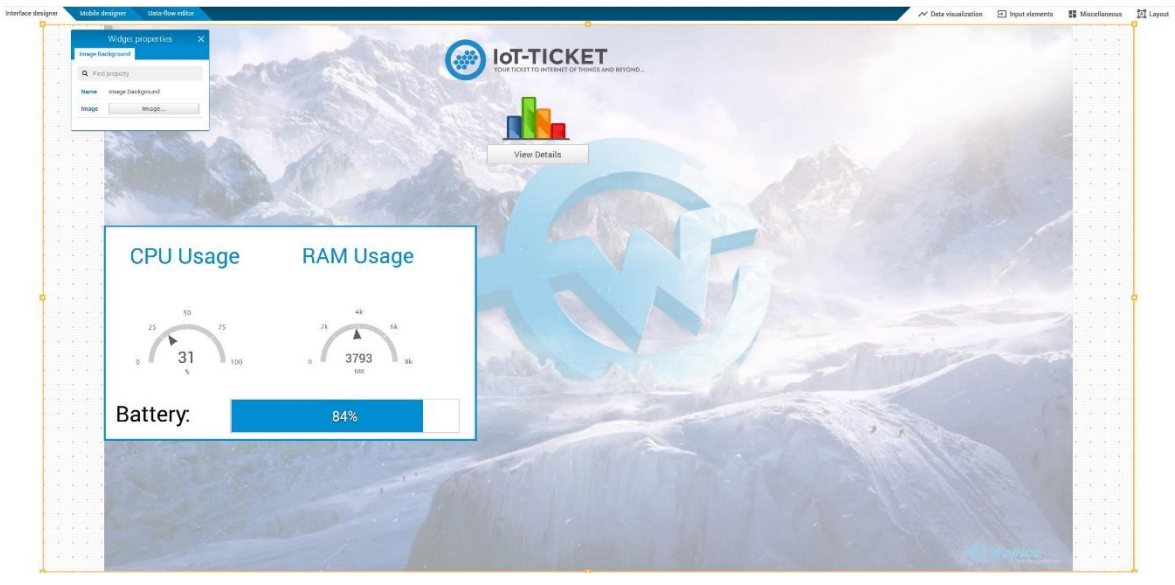
Tässä luvussa käsitellään Wapicen IoT-TICKET-tuotetta, jonka lisäomaisuudeksi tässä työssä tehty sovellus tuli. Luvussa käsitellään myös versionhallintaa ja ohjelmoinnissa yleisesti laadittuja dokumentteja.

3.1 Wapice IoT-TICKET

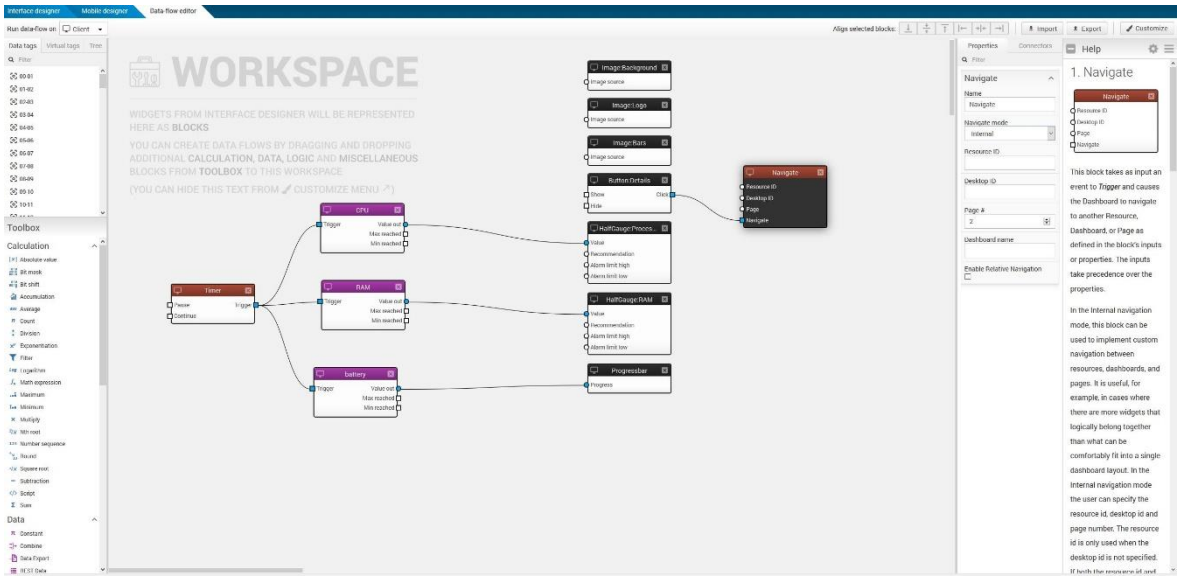
Tässä luvussa käsitellään Wapicen IoT-Ticket-tuotetta. Luvussa käsitellään tuotetta yleisesti sekä perehdytään sen eri yhteyksiin.

3.1.1 Yleistä

Wapice IoT-TICKET on suomalainen selaimella käytettävä IoT-alusta. Tuotteella voidaan luoda dashboardeja, joilla voidaan esimerkiksi ohjata kytkettyjä laitteita ja tarkistaa tehtaan reaaliaikainen tila. Dashboardin widgetit luodaan vedä ja pudota -periaatteella, joten ohjelmointi on todella helppoa. Tieto kytketään widgeteihin myös vedä ja pudota -periaatteella. (IoT-TICKET-alusta [Viitattu 31.7.2019].) Kuvassa 11 on interface designer ja kuvassa 12 on dataflow editor.

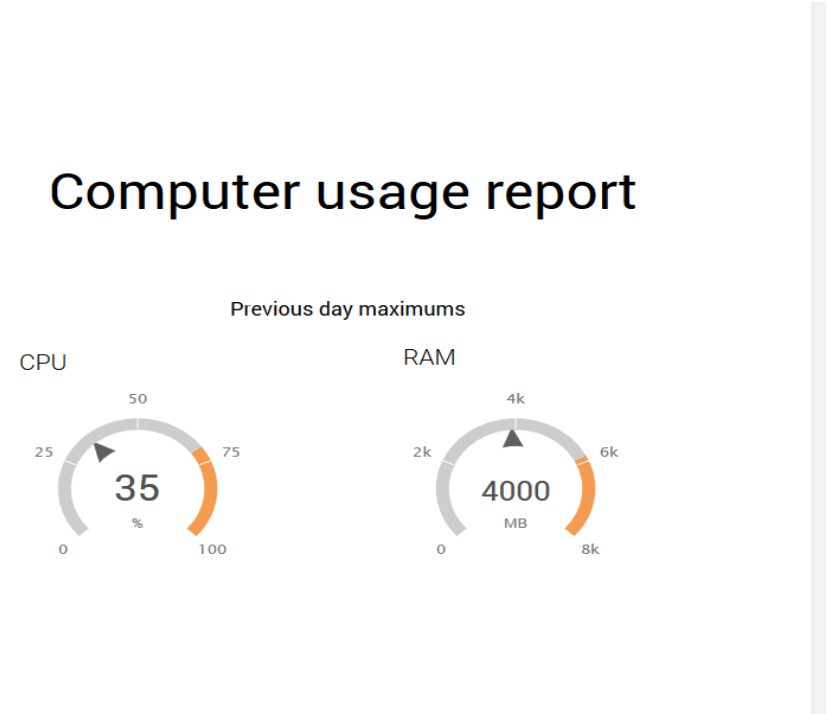


Kuva 11. IoT-TICKET interface designer



Kuva 12. IoT-TICKET dataflow editor

Alustalla on myös helppo luoda erilaisia raportteja samaan tapaan kuin tekstinkäsittelysovelluksessa. Raporttiin on myös mahdollista liittää tietoa samaan tapaan kuin dashboardeissa. Raportit ovat kuin Dashboardien ja tekstinkäsittelyn yhdistelmä. (IoT-TICKET -alusta [Viitattu 31.7.2019].)

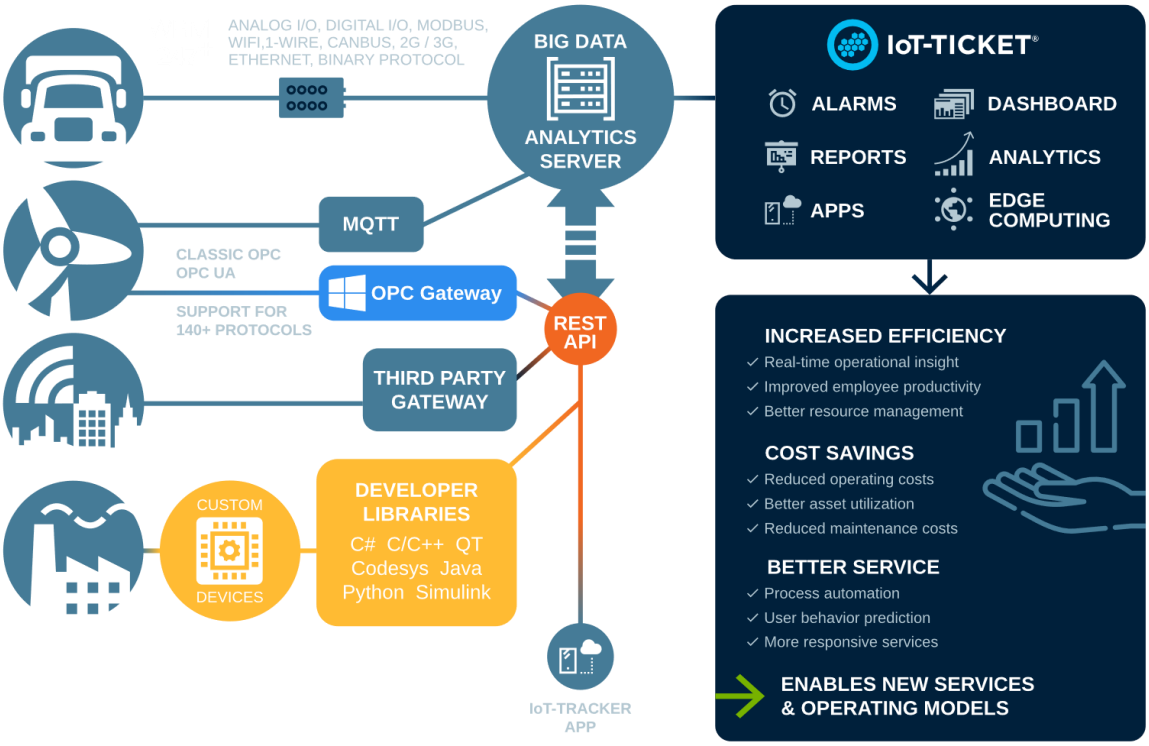


Kuva 13. IoT-TICKET-raportti

IoT-TICKET-tuotteessa voi myös tehdä erilaisia logiikoita, jotka aiheuttavat hälytyksiä. Hälytykset näkyvät IoT-TICKET-alustassa. Hälytykset on myös mahdollista lähettää sähköpostilla tai tekstiviestillä. IoT-TICKET-alustaa voi käyttää palveluna tai sen voi ottaa myös käyttöön omalta palvelimelta. (IoT-TICKET-alusta [Viitattu 31.7.2019].)

3.1.2 Yhteydet

IoT-TICKET-alustaan on mahdollista tuoda dataa Wapicen WRM247+-laitteella, MQTT tai REST API -protokollaa käyttäen (kuva 14). Kaikki yhteydet käyttävät HTTPS-protokollaa, joten liikenne on aina salattua. (IoT-TICKET api manual [Viitattu 31.7.2019], 3).



Kuva 14. IoT-TICKET -tiedonkeruu (IoT-TICKET -alusta [Viitattu 31.7.2019]).

IoT-TICKET tukee yli 140:tä protokollaa. Taulukossa 2 on esitetty kaikki tuetut protokollat.

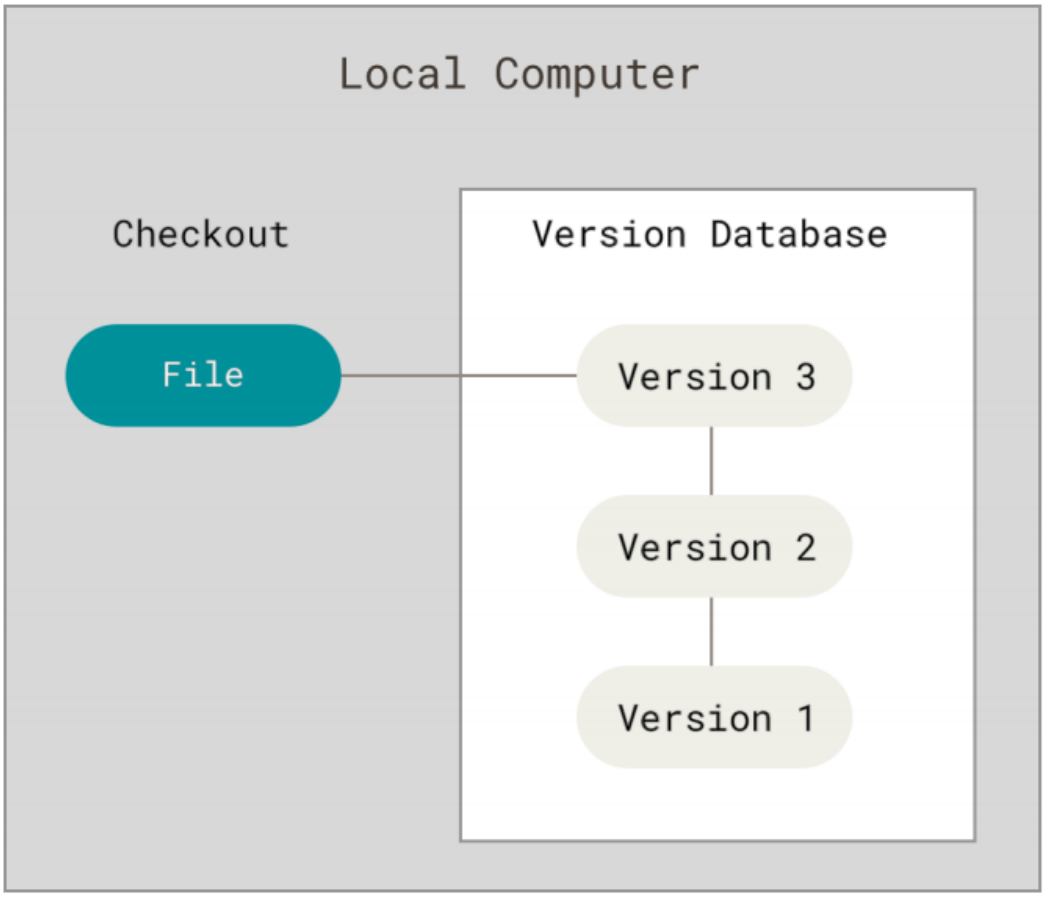
Taulukko 2. IoT-TICKET-protokollat (IoT-TICKET-alusta [Viitattu 31.7.2019]).

@aGlance	ABB Advant IMS	ABB RP570
ABB Totalflow	Air Liquide ADACS	Allen Bradley PLCs
Amaco Amocams	Anybus	Anybus-PCI
APACS (API)	APACS (Direct)	Applikon
AS-I	Aspentech (Setpoint) Setcim	AspenTech Cim/21
AspenTech InfoPlus.21	Autocom	BACnet
Bailey (Direct)	Bailey (SemAPI)	Bristol Babcock OpenBSI
CAN Data	CANopen	CDC Type 2
Citect	Databases (ODBC, MySQL, MSSQL, and Oracle)	Direct Data Exchange (DDE)
DLMS	DNP3	DSfG Data Protocol
EDAS (HOSE)	eDNA	Encore RMS
Eurotherm 800 Series	Fisher ROC (ROC & ROC Plus)	Foxboro
Foxboro I/A (Object Manager)	FoxSCADA	GCOM ABB
GE CIMPLICITY	GE Fanuc PLC	GE Harris
GE Speedtronic Mark V & VI (GSM)	GE Speedtronic Mark V (Direct)	Gensym G2
HART	Honeywell HC900	Honeywell Measurex
Honeywell OptiVISION	Honeywell PHD	Husky Host
ICCP	IEC 60870-5	IEC 61400-25
IEC 61850	Intellution Fix/iFix	Intellution Fix/iFix
ISO 11783	JagXtreme	JC N1 (Johnson Controls)
JC N2 (Johnson Controls)	Kaye Digilink	Kaye Netpac
KNX	Koyo (Direct Net) PLCs	Libelium Meshlium
Libelium Plug&Sense	LonWorks	LoRa
Lufkin Modbus	Macroview	Mark VI (Direct)
Matrikon GenCS	M-Bus	Mettler Toledo
Microsoft Task Manager	Mitsubishi PLCs	Mitsubishi Turbine Controllers
Modbus	Modbus TCP	Moore MYCRO
Motorola IP Gateway (MOSCAD)	Nanoscanner	NDC Pronet
NEG Micon	Nova Biomedical	Nova BioProfile FLEX Analyzer
OBD	ODBC	OECD
OMNI Flow Computers	OMRON PLCs	One-wire
OPC Genie	OPTO 22	Profibus
Proficy (iHistorian)	Profinet	PROMORE FibreNet DTS
Provox (CHIP)	Provox (Direct)	Quindar
Reynolds Equipment	RM-80 for Radiation Monitoring Systems	RMV9000
Roibox	RS3 RNI	Sargent/ASSAY Abloy Locks
Sartorius	SCADA Modbus	Schneider Electric PowerLogic SMS
Schneider UNI-Telway	SCI	Sensa Sensor Highway
Siemens LSX	Siemens MPI	Siemens PPI
Siemens S7 PLC for Siemens PLCs	Siemens SIMATIC TI505	Siemens Teleperm XP via XU
Siemens Wind Turbines	SNMP	SNMP
SNMP Agent	StreamInsight	Teltonika
Triconex	Triconex Achilles Certified	Triconex HPKS Certified
TVA DatAWare	ULMA	Unitrol 5000
Verano (HP) RTAP	Vestas Wind Turbine Controllers	West Series 3010 Digital Indicators
Westinghouse WDPF	WITS	WITSML
Wonderware Historian (InSQL)	Wonderware InTouch	Yokogawa
Yokogawa Vnet/IP	ZigBee	OPC DA
OPC UA Server	OPC UA Client	

3.2 Versionhallinta

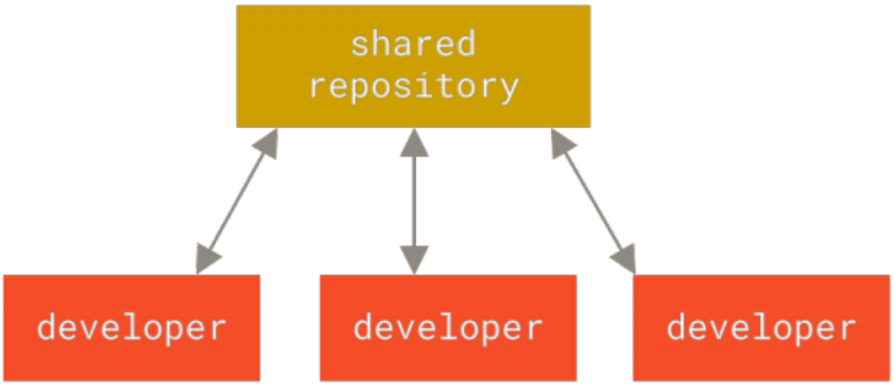
Versionhallintaan voidaan tallentaa tiedostoja. Jokaisella tallennuskerralla luodaan uusi versio. Jokaiseen luotuun versioon on mahdollista palata myöhemmin. Versionhallinnasta onkin suuri hyöty, jos tehdyt muutokset rikkovat toimivan ohjelman. Edellinen toimiva versio voidaan uudelleen ottaa käyttöön. (Chacon & Straub 2019, 9.)

Versionhallinta voidaan toteuttaa paikallisesti, keskitetysti ja hajautetusti. Paikallisessa versionhallinnassa tiedot tallennetaan paikallisen tietokoneen kiintolevylle käyttäen erikoisformaattia. (Chacon & Straub 2019, 9-10.)



Kuva 15. Paikallisen versiohallinnan rakenne (Chacon & Straub 2019, 10).

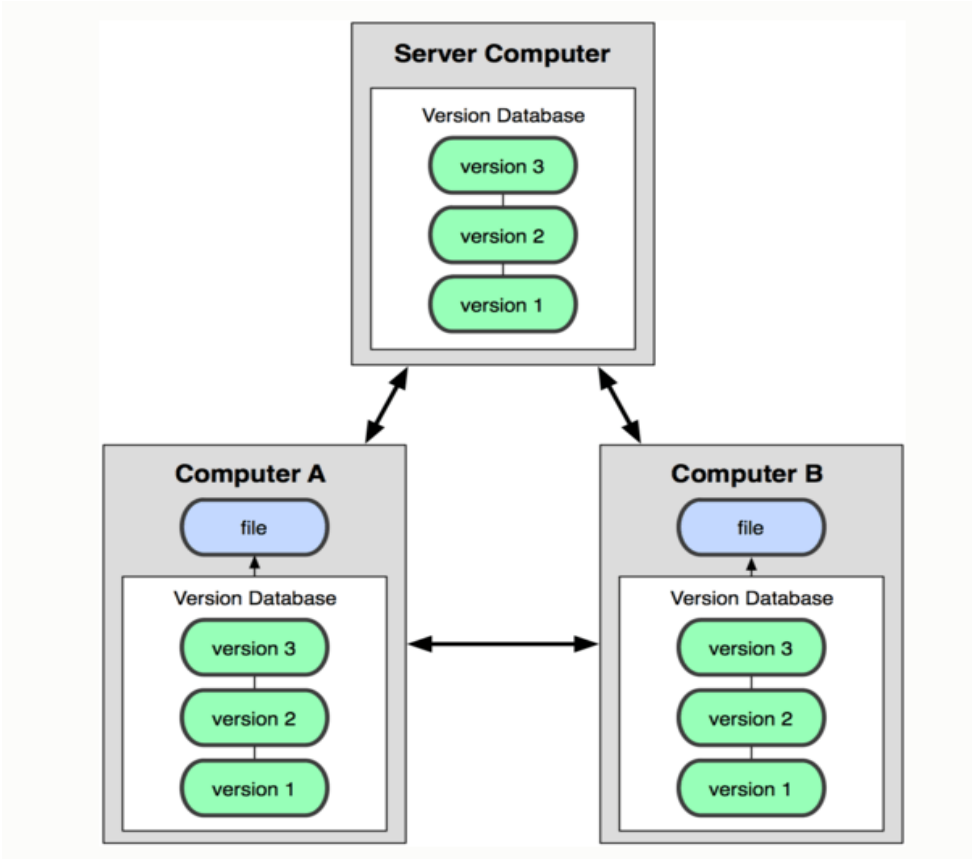
Keskitetyssä versionhallinnassa on keskitetty palvelin, missä asiakkaat voivat käyttää samaa versionhallintaa. Keskitetty versionhallinta mahdollistaa sen, että yhteistyö muiden kehittäjien kanssa onnistuu, vaikka he olisivat eri järjestelmässä. (Chacon & Straub 2019, 10-11.)



Kuva 16. Keskitetyn versiohallinnan rakenne (Chacon & Straub 2019, 11).

Hajautetussa versionhallinnassa asiakkaat eivät hae pelkästään viimeisintä versiota, vaan koko tietolähteen. Tämä ratkaisee ongelman, joka keskitetyssä versionhallinnassa on: riippuvuus yhdestä tietokannasta. Jos keskitetyn palvelimen kiintolevy korruptoituu tai hajoaa, katoaa kaikki tieto. Hajautetussa versionhallinnassa jokaisella asiakkaalla on varmuuskopio kaikesta datasta. Jos palvelimen koneen kiintolevy hajoaa, kaikki data on silti tallessa. (Chacon & Straub 2019, 11-12.)

Git on hajautettuun versionhallintaan perustuva versionhallintaohjelmisto. Git on nykyään suosittu versionhallintaohjelma. Koska Git perustuu hajautettuun versionhallintaan, sitä on nopea käyttää. Sen lisäksi on paljon operaatioita, joita voi tehdä ilman internetyhteyttä. (Chacon & Straub 2019, 13-14.)



Kuva 17. Hajautetun versiohallinnan rakenne

3.3 Dokumenttien laadinta

Ohjelmiston suunniteluun kuuluu paljon dokumentteja. Seuraavassa on esitetty tässä työssä laaditut dokumentit.

3.3.1 Vaatimusmäärittely

Vaatimusmäärittelyssä määritellään ohjelmiston tavoitteet ja vaatimukset. Siinä määritellään sekä toiminnalliset ja ei-toiminnalliset vaatimukset. Vaatimusmäärittelyssä ei oteta vielä kantaa ohjelmiston arkkitehtuuriin. (Haikala & Märijärvi 2000, 66-69.) Esimerkki toiminnallisen määrittelyn sisällöstä esitetään kuvassa 18.

1. Johdanto
    1.1 Tarkoitus
    1.2 Tuote
    1.3 Määritelmät, termit ja lyhenteet
    1.4 Viitteet, muut tähän liittyvät dokumentit
    1.5 Yleiskatsaus dokumenttiin
2. Yleiskuvaus
    2.1 Ympäristö
    2.2 Toiminta
    2.3 Käyttäjät
    2.4 Yleiset rajoitteet
    2.5 Oletukset ja riippuvuudet
3. Tiedot ja tietokanta
    3.1 Tietosisältö
    3.2 Käyttöintensiteetti
    3.3 Kapasiteettivaatimukset
    3.4 Tiedostot ja asetustiedostot
4. Toiminnot
    4.n Toiminnon kuvaus
5 Ulkoiset liittymät
    5.1 Käyttöliittymä
    5.2 Laitteistoliittymät
    5.3 Ohjelmistoliittymät
    5.4 Tietoliikenneliittymät
6 Muut ominaisuudet
    6.1 Suorituskyky
    6.2 Käytettävyys, toipuminen, turvallisuus ja suojaukset
    6.3 Ylläpidettävyys
    6.4 Siirrettävyys, yhteensopivuus
    6.5 Operointi
7 Suunnittelurajoitteet
    7.1 Standardit
    7.2 Laitteistorajoitteet
    7.3 Ohjelmistorajoitteet
    7.4 Muut rajoitteet

Kuva 18. Esimerkki toiminnallisen määrittelyn sisältörungosta (Haikala & Märijärvi 2000, 68).

3.3.2 Tekninen määrittely

Teknisessä määrittelyssä määritellään ohjelmiston arkkitehtuuri. Siihen kuuluvat muun muassa seuraavat asiat: tietokanta-arkkitehtuuri, ohjelmistoarkkitehtuuri, moduulit ja prosessit. (Haikala & Märijärvi 2000, 69-72.) Esimerkki teknisen määrittelyn sisältörungosta esitetään kuvassa 19.

```
1. Johdanto
        1.1 Tarkoitus
        1.2 Dokumentin kattavuus
        1.3 Määritelmät, termit ja lyhenteet
        1.4 Viittaukset muihin dokumentteihin, standardeihin, käsikirjoihin jne...
        1.5 Yleiskatsaus dokumenttiin
2. Järjestelmän yleiskuvaus
        Sovellusalueen kuvaus, järjestelman osuus siinä, laitteisto- ja
        ohjelmistoympäristön kuvaus, toteutuksen keskeiset reunaehdot ja
        järjestelmän liittyminen ympäristöönsä
3. Arkkitehtuurin kuvaus
        3.1 Ratkaisuperiaatteet
        3.2 Tietokanta-arkkitehtuuri
        3.3 Ohjelmistoarkkitehtuuri: Moduulit ja prosessit.
        3.4 Uudelleenkäytettävät komponentit
4. Moduuli (ja prosessi) -kuvaukset
        4.i Kustakin moduulista (ja prosessista) esitetään omassa luvussaan
                yleiskuvaus
                attribuutit (ylläpidettävät tilatiedot)
                operaatiot (rajapinnan määrittely)
                poikkeus- ja virhetilanteiden käsittely (myös "odottamattomien") sekä
                ohjeita moduulisuunnittelua ja toteutusta varten (viittaukset käytettäviin
                algoritmeihin, uudelleenkäytettävyys. jne.)
Muita mahdollisia kohtia
        Ylläpito-ohjeet
        Siirrettävyys
        Virhetilanteiden käsittely
        Luotettavuus
        Erityiset tekniset ratkaisut
        Hylättyjä ratkaisuvaihtoehtoja, miksi ne hylättiin
        Käyttöliittymä. jos ei tarkasti määrittelyssä
        Ratkaisun rajoitteet
        Testattavuus
        Jäljitettävyys
```

Kuva 19. Esimerkki teknisen määrittelyn sisältörungosta (Haikala & Märijärvi 2000, 72).

3.3.3 Testaussuunnitelma

Testaussuunnitelmassa kerrotaan mm. mitä testejä tehdään, milloin ja minkälaisia tuloksia odotetaan (Haikala & Märijärvi 2000, 281-282).

```
1. Johdanto
2. Testauksen kohde ja tavoitteet
3. Testausympäristö
4. Testauksen organisointi ja raportointi
5. Testausstrategia ja integrointisuunnitelma
6. Testattavat toiminnot
7. Toimintojen testitapaukset, hyväksymiskriteerit
8. Ei-toiminnallisten ominaisuuksien testaus
9. Erikoistilanteet
10. Ominaisuudet, joita ei testata
```

Kuva 20. Esimerkki testaussuunnitelman sisällöstä (Haikala & Märijärvi 2000, 282).

4 SOVELLUKSEN TOTEUTUSVAIHEET

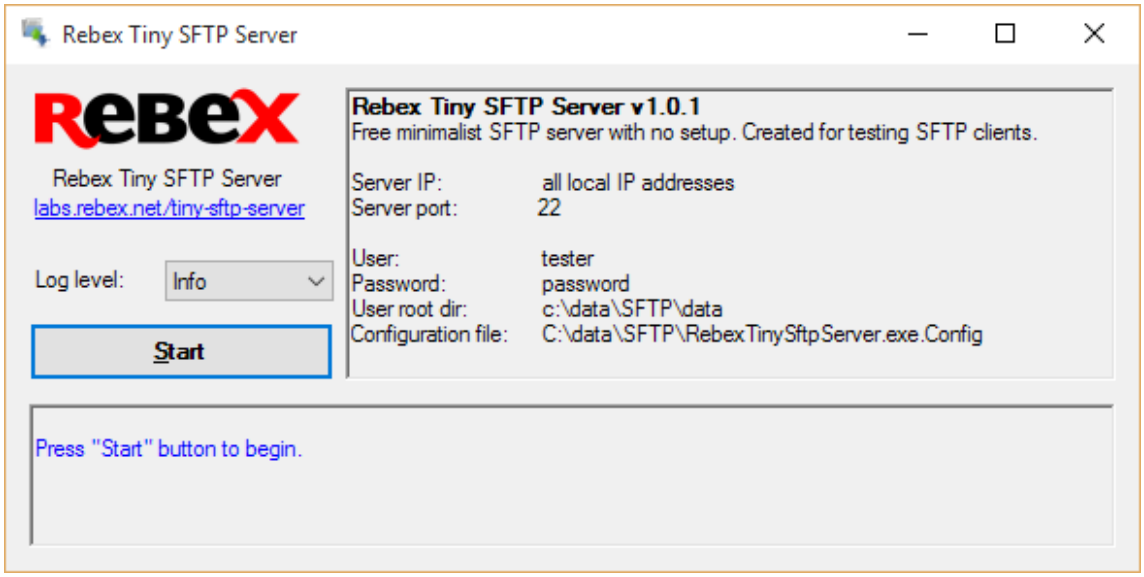
Tässä luvussa käydään läpi sovelluksen toteutukseen liittyvät vaiheet aina datan simuloinnista sovelluksen ohjelmointiin.

4.1 Datan simulointi

Sähkön hintatietojen saamiseksi Nord Poolilta olisi tarvittu kallis lisenssi. Tämän vuoksi työssä käytettiin simuloitua tietoa. Tämä toteutettiin luomalla Microsoft Excel -tiedosto omalle SFTP-palvelimelle. Excel-tiedosto on hyvin samanlainen, kuin Nord Poolilla on tarjolla. Näin toteutettuna muutos sovellukseen oikean datan saamiseksi on nopea ja helppo.

Nord Poolilta on saatavilla Day-Ahead-hinnat eli seuraavan päivän hintatiedot. Yhden päivän ajalta luotiin hinnat, joita käytettiin demonstraatiotarkoituksessa. Tässä työssä käytettiin käyttäjätunnusta ja salasanaa kirjautumiseen SFTP-palvelimelle. SFTP-palvelimille on mahdollista kirjautua myös käyttäen RSA-avainta. Mikäli käyttäjätunnusta ja salasanaa ei ole annettu, ohjelma käyttää tunnistautumiseen RSA-avainta.

SFTP- palvelimena toimi Rebex Tiny SFTP Server, joka oli helppo ottaa käyttöön ja ominaisuuksia oli riittävästi tätä käyttötarkoitusta varten.



Kuva 21. Rebex Tiny SFTP Server

4.2 Sovelluksen ohjelmointi

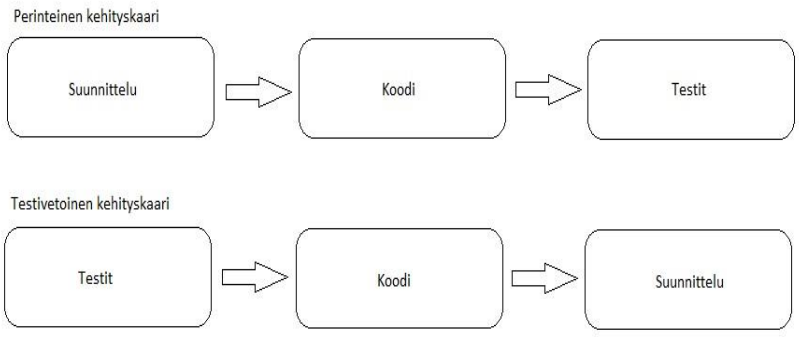
Ohjelmoinnissa käytettiin ohjelmointiympäristönä Eclipse-ohjelmistoa. Eclipse IDE on avoimen lähdekoodin kehitysympäristö, joka tukee muun muassa Java-, C/C++- ja PHP -ohjelmointikieliä (Eclipse [Viitattu 10.8.2019]).

Sovellus toteutettiin Java Spring Boot -kielellä. Java Spring Bootiin päädyttiin seuraavien ominaisuuksien takia: aikaisempi kokemus Spring Bootista, kielellä on helppoa luoda REST-rajapintoja ja mikropalveluominaisuuksien helppo käyttöönotto.

Sovelluksen lähdekoodin tallennukseen ja versionhallintaan käytettiin Atlassianin BitBucket Git -ratkaisua. Git-ratkaisun avulla pystyi helposti palaamaan vanhaan versioon, jos uudessa versiossa ilmeni ongelmia.

4.2.1 Testit

Sovellus kehitettiin käyttäen testivetoista kehitystä. Testivetoisessa kehityksessä kirjoitetaan ensin testit ja vasta sen jälkeen varsinainen ohjelma, jonka pitää läpäistä testit. Käytetyimmät testit ovat yksikkö- ja integraatiotestit (Koskela 2009, 4).



Kuva 22. Sovelluksen kehityskaaret

Yksikkötestissä testataan pienintä mahdollista ohjelman osaa. Yksikkötestissä testattavan osan tulisi tehdä vain yksi asia. Testivetoisessa kehityksessä yksikkötestit ovat tärkeässä roolissa. (Rouse 2017.)

```
14⊖    @Test
15     public void test() {
16         assertEquals(priceInfo.getTime(),"22");
17         assertEquals(priceInfo.getPrice(),"21.5");
18     }
19
```

Kuva 23. Koodia yksikkötestistä

Integraatiotestissä testataan useamman komponentin toimintaa yhdessä. Esimerkiksi koko ohjelmiston toiminnan testaus on integraatiotesti. Vaikka kaikki yksikkötesteissä testatut komponentit menisivät läpi moitteetta, integraatiotesteissä saattaa silti tulla ongelmia. Saattaa olla, että komponentit eivät toimi oikein, kun ne on yhdistetty toisiinsa. (Intergration Testing [Viitattu 8.8.2019].)

4.2.2 Mikropalvelu

Ohjelmiston tekeminen aloitettiin mikropalvelun mahdollistavien ominaisuuksien lisäämisellä. Mikropalvelusovelluksessa tarvitaan instanssipalvelin, josta löytyy tiedot kaikista siihen rekisteröityneistä instansseista. Mikropalvelun pitää myös rekisteröidä itsensä instanssipalvelimelle. Spring Bootissa tämä on helppo toteuttaa. Seuraavassa kuvassa 24 esitetään yksi tapa, kuinka toteuttaa mikropalvelun rekisteröinti Eureka-instanssipalvelimeen:

Aluksi täytyy määrittää seuraavat riippuvuudet pom.xml-tiedostossa: spring-cloud-started-config ja spring-cloud-starter-netflix-eureka-client. Tämän jälkeen lisätään annotaatio @EnableEurekaClient

```
17 @SpringBootApplication
18 @EnableEurekaClient
19 public class IntegrationApplication {
20
21     public static void main(String[] args) {
22         SpringApplication.run(IntegrationApplication.class, args);
23
24     }
25
26 }
27
```

Kuva 24. Eureka Client -annotaatio

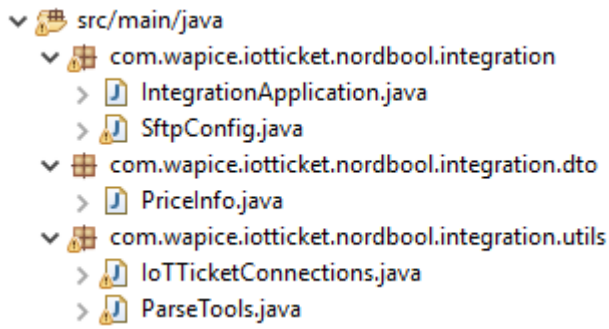
Kun vielä lisätään Eureka-palvelimen tiedot konfiguratioon, mikropalvelu rekisteröi itsensä Eureka-palvelimelle ja on käytettävissä.

```
19 #Eureka config
20 # Service name
21 spring.application.name=nordbool
22 # Port
23 server.port=8200
24 # Eureka server url
25 eureka.client.service-url.default-zone=http://localhost:8761/eureka
26
```

Kuva 25. Esimerkki Eureka-konfiguraatiosta

4.2.3 Luokat

Ohjelmaa tehtäessä käytettiin olio-ohjelmoinnin periaatteita ja ohjelma jaettiin seuraaviin luokkiin: IntegrationApplication, SftpConfig, PriceInfo, IoTTicketConnections ja ParseTools.



Kuva 26. Luokat ja paketit

IntegrationApplication sisältää Main-metodin, joka tarvitaan ohjelman käynnistämiseen. SftpConfig sisältää SFTP-palvelimelta lataamista varten tarvittavat konfiguraatiot. Sieltä kutsutaan myös parsinta ja lähetysmetodeja tiedoston lataamisen jälkeen. PriceInfo sisältää hintatieto- ja kellonaika-jäsenmuuttujat. IoTTicketConnections sisältää kaikki metodit, joita tarvitaan datan lähettämiseksi IoT-TICKET-alustaan. ParseTools sisältää metodit, joita tarvitaan hintatietojen keräämiseen tiedostosta.

4.2.4 Excel-tiedoston lataus

Varsinainen ohjelmointi aloitettiin sillä, että otettiin yhteys omalla koneella käynnissä olevaan SFTP-palvelimeen ja ladattiin siellä oleva Excel-tiedosto. Tämä toteutettiin käyttämällä Spring-moduulia org.springframework.integration.sftp.

Aluksi asetettiin SFTP session factory, jossa asetettiin tarvittavat parametrit: osoite, portti, käyttäjätunnus ja salasana.

```
@Bean
public SessionFactory<LsEntry> sftpSessionFactory() {
    DefaultSftpSessionFactory factory = new DefaultSftpSessionFactory(true);
    factory.setHost(sftpHost);
    factory.setPort(sftpPort);
    factory.setUser(sftpUser);
    if (sftpPrivateKey != null) {
        factory.setPrivateKey(sftpPrivateKey);
        factory.setPrivateKeyPassphrase(sftpPrivateKeyPassphrase);
    } else {
        factory.setPassword(sftpPasword);
    }
    factory.setAllowUnknownKeys(true);
    return new CachingSessionFactory<LsEntry>(factory);
}
```

Kuva 27. Esimerkki SFTP session factory -asetuksesta

Tämän lisäksi täytyi määrittää bean MessageSource<> ja määrittää se @InBoundChannelAdapter-annotaatiolla. Tuossa beanissa määritellään myös cron-ajastus, joka määrittää sen, kuinka usein tiedosto käydään lataamassa SFTP-palvelimelta. Lopuksi luodaan messageHandler, jossa käsitellään ladattu tiedosto.

4.2.5 Excel-tiedoston käsittely

Excel-tiedostosta etsittiin tarvittavat kentät. Tätä varten käytettiin org.apache.poi.hssf- ja org.apache.poi.ss-moduuleja. Edellä mainittujen moduulien avulla tiedostosta haettiin halutut arvot ja tallennettiin listaan PriceInfo-oliona.

Excel-solujen arvoissa saattaa olla jotain erikoismerkkejä, jotka eivät ole näkyvissä esim. \n. Näistä aiheutuvien ongelmien vuoksi täytyy käyttää Regex-toimintoa, jolla saadaan poistettua kaikki muut merkit, paitsi kirjaimet ja numerot.

4.2.6 Hintatietojen lähetys IoT-TICKET-palveluun

Haetut hinnat täytyi vielä lähettää IoT-TICKET-palveluun halutuille palvelun tilanneille asiakkaille. Lähettämiseen käytettiin IoT-TICKET-järjestelmän julkista REST-API Java -moduulia. Moduuli lähettää REST-API:n kautta viestit IoT-TICKET-järjestelmän tietokantaan.

Aluksi hintatiedoista luodaan DatanodeWriteValue-olioita.

```
public Datanode.DatanodeWriteValue CreateDatanodeWriteValue(String name, String path, String unit, Object value) {
    Long currTime = (Calendar.getInstance(TimeZone.getTimeZone("GMT"))).getTimeInMillis();
    Datanode.DatanodeWriteValue dnwrite = new Datanode.DatanodeWriteValue();
    dnwrite.setName(name);
    if(!path.isEmpty())
        dnwrite.setPath(path);
    dnwrite.setUnit(unit);
    dnwrite.setValue(value);
    dnwrite.setTimestampMilliseconds(currTime);
    return dnwrite;
}
```

Kuva 28. Metodi DatanodeWriteValuen luontiin

Tämän jälkeen oliot lisättiin listaan ja ne lähetettiin moduulin writeData-metodin avulla Iot-TICKET-järjestelmään.

5 TULOKSET

Sovellus latasi SFTP-palvelimelta Excel-tiedoston, etsi siitä hintatiedot ja lähetti ne eteenpäin IoT-TICKET-järjestelmään datanodeina. Sovellus suoritti nämä tapahtumat päivittäin tiettyyn aikaan.

Lopputuloksena oli hyvin toimiva sovellus, joka kesti virhetilanteita hyvin. Sovellukselle kirjoitetut testit menivät läpi ongelmitta. Aluksi työtä suunnitellessa ajatuksena oli tehdä sovelluksesta mikropalvelusovellus. Suunnitelmien edetessä päädyttiin ratkaisuun, että sovelluksesta tehdään monoliittisovellus. Tässä tilanteessa se oli parempi ratkaisu. Tuotteesta saadaan helposti muokattua tarvittaessa mikropalvelusovellus.

Sovellusta ei ole tällä hetkellä vielä otettu käyttöön, mutta sovellus on kuitenkin täysin käyttövalmis. Jatkokehityksenä asiakaskonfiguraation hallintaa voisi kehittää helppokäyttöisemmäksi ja testejä voisi kirjoittaa lisää.

Työ opetti paljon ohjelmoinnista ja siitä on varmasti hyötyä jatkossa. Erityisesti mikropalveluista saatu oppi on hyödyksi, koska tulevaisuudessa tehdään varmasti paljon mikropalveluarkkitehtuuriin perustuvia sovelluksia.

6 YHTEENVETO JA POHDINTA

Tämän opinnäytetyön tavoite oli tehdä Wapicen IoT-TICKET-tuotteeseen lisäominaisuus, joka hakee sähkön pörssihinnat Nord Poolin SFTP-palvelimelta, kerää hinnat tiedostosta ja lähettää ne IoT-TICKET-järjestelmään palvelun tilanneille asiakkaille. Opinnäytetyön lähteinä toimivat kirjat ja internet.

Työssä tarkasteltiin mikropalveluarkkitehtuuria, sekä muita ohjelmointiin liittyviä toimintatapoja ja dokumentteja. Työssä käsiteltiin myös ohjelmistojen testausta.

Sovelluksen ohjelmointi oli helppoa, sillä kaikkiin toimintoihin löytyi Javasta ja Spring Bootista valmiita komponentteja. SFTP-yhteyksiin, Excel-tiedoston parsintaan ja IoT-TICKET-järjestelmän REST API -rajapintaan löytyi valmiit kirjastot. Myös erilaisiin mikropalvelutoimintoihin Spring Cloudista löytyi valmiita moduuleita. Opinnäytetyön edessä huomattiin, että sovelluksesta ei ole tarvetta tehdä mikropalvelua, joten se osio jätettiin ohjelmasta pois. Tulevaisuuden tarpeiden mukaan ohjelmistokomponentti voidaan jatkokehityksenä integroida tiukemmin IoT-TICKET-järjestelmän sisälle.

Suurin haaste työssä oli käyttäjäoikeuksien järkevä ja turvallinen toteutus. Nämä saatiin kuitenkin toteutettua järkevästi ja turvallisesti.

LÄHTEET

Chacon, S. & Straub, B. 2019. Pro Git. [Verkkokirja]. [Viitattu 16.8.2019]. Saatavilla https://github.com/progit/progit2/releases/download/2.1.161/progit.pdf

Collin, J. & Saarelainen, A. 2016. Teollinen Internet. [Verkkokirja]. Helsinki: Talentum. [Viitattu 30.7.2019]. Saatavana Alma Talent bisneskirjasto -tietokannasta. Vaatii käyttöoikeuden.

Eclipse. Ei päiväystä. Eclipse IDE. [Verkkosivu]. [Viitattu 10.8.2019]. Saatavilla: https://www.eclipse.org/eclipseide/

Governor, J. 2017. The race to own the pipeline. [Blogi-kirjoitus]. [Viitattu 23.8.2019]. Saatavilla https://www.enterpriseirregulars.com/116202/race-pipeline-atlassian-aint-playin-introducing-devops-marketplace/

Haikala, I. & Märijärvi, J. 2000. Ohjelmistotuotanto. Helsinki: Talentum.

Integration testing. Ei päiväystä. Integration Testing. [Verkkosivu]. Techopedia. [Viitattu 8.8.2019]. Saatavilla https://www.techopedia.com/definition/7751/integration-testing

Introduction to Hystrix. 2019. Introduction to Hystrix. [Verkkosivu]. [Viitattu 9.8.2019]. Saatavilla: https://www.baeldung.com/introduction-to-hystrix

Introduction to Spring Framework. Ei päiväystä. Introduction to Spring Framework. [Verkkosivu]. [Viitattu 9.8.2019]. Saatavilla https://www.geeksforgeeks.org/introduction-to-spring-framework/

IoT-TICKET alusta. Ei päiväystä. IoT-TICKET alusta. [Verkkosivu]. [Viitattu 31.7.2019]. Saatavilla: https://www.iot-ticket.com/fi/alusta

IoT-TICKET api manual. Ei päiväystä. IoT-TICKET REST API MANUAL. [Verkkojulkaisu]. [Viitattu 31.7.2019]. Saatavilla: https://iot-ticket.com/images/Files/IoT-Ticket.com_IoT_API.pdf

Klemetti, M. 2013. Mitä on DevOps. [Blogi-kirjoitus]. [Viitattu 23.8.2019]. Saatavilla https://www.eficode.com/blogi/blogi/mita-on-devops

Kornilova, I. 28.4.2017. DevOps is a culture not a role. [Blogi-kirjoitus]. [Viitattu 9.8.2019]. Saatavilla https://medium.com/@neonrocket/devops-is-a-culture-not-a-role-be1bed149b0

Koskela, L. 2009. Test driven. Greenwich: Manning publications co.

Koskimies, K. 2000. Oliokirja. Helsinki: Talentum.

Nord Pool. Ei päiväystä. Nord Pool homepage. [Verkkosivu]. Norway: Nord Pool AS. [Viitattu 7.8.2019]. Saatavilla https://www.nordpoolgroup.com/

Nord Pool group. Ei päiväystä. Nord Pool – About us. [Verkkosivu]. Norway: Nord Pool AS. [Viitattu 7.8.2019]. Saatavilla https://www.nordpoolgroup.com/About-us/

Rajesh, R. 2016. Spring Microservices. [Verkkokirja]. Birmingham: Packt Publishing. [Viitattu 20.7.2019]. Saatavana Ebsco eBook Collection -tietokannasta. Vaatii käyttöoikeuden.

Rouse, M. 2017. Unit testing. [Verkkosivu]. [Viitattu 7.8.2019]. Saatavana: https://searchsoftwarequality.techtarget.com/definition/unit-testing

Spring Boot. Ei päiväystä. Spring Boot – Introduction. [Verkkosivu]. [Viitattu 9.8.2019]. Saatavilla https://www.tutorialspoint.com/spring_boot/spring_boot_introduction.htm

Spring Framework. Ei päiväystä. Spring Framework – Overview. [Verkkosivu]. [Viitattu 9.8.2019]. Saatavilla https://www.tutorialspoint.com/spring/spring_overview

Wallenius, N. Ei päiväystä. Mitä mikropalvelut ovat [Blogi-kirjoitus]. [Viitattu 9.8.2019]. Saatavilla https://niklaswallenius.fi/teknologiat/mikropalvelut/

Wapice. Ei päiväystä. Wapice – Software, Electronics, Innovation. [Verkkosivu]. [Viitattu 31.7.2019]. Saatavilla: https://www.wapice.com/fi

What is Micro Service. Ei päiväystä. What is Micro Service. [Verkkosivu]. [Viitattu 9.8.2019]. Saatavilla https://www.optisolbusiness.com/insight/micro-services-architecture-spring-boot-and-netflix-infrastructure